

Nucleus NET Extended Protocol Package

Reference Manual



0001076-001 Rev. 101

Copyright (c) 2001
Accelerated Technology, Inc.
720 Oak Circle Dr. E.
Mobile, AL 36609
(251) 661-5770



Related Documentation

Nucleus PLUS Reference Manual, by Accelerated Technology, describes the operation and usage of the Nucleus PLUS kernel.

Nucleus PLUS Internals, by Accelerated Technology, describes, in considerable detail, the implementation of the Nucleus PLUS kernel.

Nucleus NET Reference Manual, by Accelerated Technology, describes the installation and operation of the TCP/IP Nucleus PLUS network facilities.

Style and Symbol Conventions

Program listings, program examples, filenames, menu items/buttons and interactive displays are each shown in a special font.

Program listings and program examples - *Courier New*

Filenames - *COURIER NEW, ALL CAPS*

Interactive Command Lines - ***Courier New, Bold***

Menu Items/Buttons – *Times New Roman Italic*

Trademarks

MS-DOS is a trademark of Microsoft Corporation

UNIX is a trademark of X/Open

IBM PC is a trademark of International Business Machines, Inc.

Intel is a trademark of Intel, Inc.

Additional Assistance

For additional assistance, please contact us at the following:

Accelerated Technology

720 Oak Circle Drive, East

Mobile, AL 36609

800-468-6853

(251)661-5770

(251)661-5788 (fax)

support@atinucleus.com

<http://www.atinucleus.com>

Copyright (©) 2001, All Rights Reserved.

Document Part Number: 0001076-001 Rev. 101

Last Revised: July 30, 2001





Contents

Chapter 1 – Introduction.....	1
Nucleus FTP Server.....	2
Nucleus FTP Client.....	2
Nucleus Telnet.....	2
Nucleus TFTP Server	2
Chapter 2 – Nucleus FTP Package	3
Files Shipped with Nucleus FTP	4
Nucleus FTP Commands Supported.....	5
FTP Common Defines and Data Structures.....	8
Defines	8
Data Structures.....	9
FTP Server-Level Defines and Data Structures.....	10
Defines	10
FTP_GET_CURRENT_YEAR	11
Data Structures.....	11
FTP Server - Simple User Authentication	12
Defines	12
Data Structures.....	13
Configurable Variable.....	13
FTP Server-Level Functions.....	13
NU_FTP_Server_Init.....	13
FTP Client-Level Defines and Data Structures	14
Defines	14
Type Definitions	14
FTP Client-Level Functions	16
NU_FTPC_Client_Open.....	16
NU_FTPC_Client_Close	17
NU_FTPC_Client_Login.....	18
NU_FTPC_Client_Get.....	19
NU_FTPC_Client_Put	20
NU_FTPC_Client_Append_To_File	21
NU_FTPC_Client_ChDir	22
NU_FTPC_Client_Dir	23



NU_FTPC_Client_Nlist.....	24
NU_FTPC_Client_Tran_Type.....	25
NU_FTPC_Client_RmDir	26
NU_FTPC_Client_Mkdir.....	27
NU_FTPC_Client_Status.....	28
NU_FTPC_Client_Rename_File	29
NU_FTPC_Client_Tran_Mode.....	30
FTP Client-Level Command Primitives	31
NU_FCP_Reply_Read	31
NU_FCP_Client_Verify_Caller.....	32
NU_FCP_Client_Tran_Ack.....	33
NU_FCP_Client_TYPE.....	34
NU_FCP_Client_MODE	35
NU_FCP_Client_STRU.....	36
NU_FCP_Client_PORT.....	37
NU_FCP_Client_LIST.....	38
NU_FCP_Client_RETR.....	39
NU_FCP_Client_STOR.....	40
NU_FCP_Client_NLST	42
NU_FCP_Client_USER.....	43
NU_FCP_Client_PASS	44
NU_FCP_Client_ACCT	45
NU_FCP_Client_CWD NU_FCP_Client_XCWD	46
NU_FCP_Client_MKD NU_FCP_Client_XMKD	47
NU_FCP_Client_RMD NU_FCP_Client_XRMD.....	48
NU_FCP_Client_CDUP NU_FCP_Client_XCUP	49
NU_FCP_Client_PASV.....	50
NU_FCP_Client_QUIT	51
NU_FCP_Client_SYST	52
NU_FCP_Client_STAT	53
NU_FCP_Client_HELP.....	54
NU_FCP_Client_PWD NU_FCP_Client_XPWD	55
NU_FCP_Client_NOOP	56
NU_FCP_Client_RNFR.....	57
NU_FCP_Client_RNTO	58
NU_FCP_Client_DELE.....	59



Chapter 3 – Nucleus Telnet.....	61
Files Shipped with Nucleus Telnet	62
Defines and Data Structures	62
Defines	62
TN_VERSION_COMP.....	62
Data Structures.....	63
Service Call Primitives	64
NU_Telnet_Server_Accept.....	64
NU_Telnet_Send	65
NU_Telnet_Start_Negotiate	66
NU_Telnet_Specific_Negotiate	67
NU_Install_Negotiate_Options.....	68
NU_Telnet_Init_Parameters	69
NU_Telnet_Free_Parameters.....	70
NU_Telnet_Check_Connection	71
NU_Telnet_Client_Connect	72
NU_Received_Exit	73
NU_Close_And_Check_Retval	74
NU_Telnet_Socket.....	75
Internal Service Calls.....	76
NU_Telnet_Pick_Up_Ascii	76
NU_Telnet_Parse.....	77
NU_Telnet_Get_Filtered_Char.....	78
NU_Receive_NVT_Key	79
NU_Send_NVT_Key.....	80
Chapter 4- Nucleus TFTP Server	81
Introduction.....	82
Files Shipped with Nucleus TFTP Server.....	82
TFTP Server Defines and Data Structures.....	83
Defines	83
Data Structures.....	84
TFTP Server Initialization	85
TFTP_Server_Init	85
Chapter 5 – Demonstration Application.....	87
Configuring Nucleus Extended Protocol Package Demonstration	88
DEMOLC.....	89
XPROTD.C.....	89
FTP Client.....	90
FTP Server	91
TFTP Server.....	91
Telnet Server.....	92



Appendix A – Enhancements and API Changes	93
Extended Protocol Package 1.2	94
Enhancements	94
Newly Defined Macros	94



1

Introduction

Nucleus FTP Server

Nucleus FTP Client

Nucleus Telnet Server

Nucleus TFTP Server



The Nucleus Extended Protocol Package contains an RFC compliant FTP Server, FTP Client, Telnet Server and TFTP Server. The demonstration program `XPROTD.C` provides a demonstration of each of the four protocols. The FTP Server, FTP Client and TFTP Server are dependent on the Nucleus File Abstraction Layer for file I/O operations.

Nucleus FTP Server

The Nucleus FTP Server API is a set of functions for use with Nucleus PLUS in conjunction with Nucleus NET. These functions provide an RFC compliant FTP Server implementation and serve as an example for building custom servers.

Nucleus FTP Client

The Nucleus FTP Client API is a set of functions for use with Nucleus PLUS in conjunction with Nucleus NET. These functions provide an RFC compliant framework upon which an FTP Client application can be built. They are designed to allow for a variety of different client implementations, ranging from an interactive user-oriented client to an embedded special-use client.

Nucleus Telnet

The Nucleus Telnet API is a set of functions for use with Nucleus PLUS in conjunction with Nucleus NET. These functions provide an RFC compliant framework upon which a simple Telnet application can be built for an embedded system. For example, a Telnet debugger server can be built as a task in an application. The Telnet facility could then be used on another computer to connect to this debugger server and retrieve Nucleus Debugger information for the application.

Nucleus TFTP Server

The Nucleus TFTP Server is an implementation of an RFC compliant TFTP server designed to run on top of the Nucleus NET protocol stack and the Nucleus PLUS RTOS. TFTP (Trivial File Transfer Protocol) is, as its name implies, a very simple file transfer protocol. Using the Nucleus TFTP Server, a user can implement the TFTP Server and have it executing as a separate task than the user's application. The Nucleus TFTP Server is compatible with RFC 2347 compliant and non-compliant TFTP Client applications.



Nucleus FTP Package

Files Shipped with Nucleus FTP

Nucleus FTP Commands
Supported

Nucleus FTP Common Defines And
Data Structures

FTP Server-Level Defines And
Data Structures

FTP Server-Level Functions

FTP Client Defines And Data
Structures

FTP Client-Level Functions

FTP Client-Level Command
Primitives

Files Shipped with Nucleus FTP

Nucleus FTP is comprised of the files shown in the table below. For specific notes on your platform, refer to the *Target Specific Notes* that came with Nucleus FTP.

File Name	Description
FTPS.C	Server-level API functions, which provide a basic FTP Server implementation.
FSP.C	Server-level command primitives needed to create an RFC compliant FTP Server.
FST.C	Server-level application initialization routines responsible for defining the initial application environment.
FTP_CFG.C	Configuration data for the FTP client and FTP server. Data structures can be customized by the application developer.
FTPC.C	Client-level API functions that provide a basic FTP Client implementation.
FC.C	Functions that can be called by both the client and the server. It helps in consolidating duplicate code.
FCP.C	Client-level command primitives needed to create an RFC compliant FTP Client application.
FC_DEFS.H	Common defines for the FTP Client and Server. It contains API constant defines as well as definitions of all API specific data structures.
FC_EXTR.H	Function prototypes and defined constants for FTP Client and Server functions. All correlating functions are defined in FC.C
FTPS_DEF.H	Definitions for the FTP Server-level API. It includes the API constant defines as well as definitions of all API specific data structures.
FSP_EXTR.H	Function prototypes for the FTP server command primitives found in FSP.C.
FST_EXTR.H	Function prototypes for the Server-level tasks and general application initialization utilities found in FST.C.
FTPS_EXT.H	This file contains the function prototypes for the Server-level API and general utilities found in FTPS.C.
FTPC_DEF.H	Definitions for the FTP Client-level API. It includes the API constant defines as well as definitions of all API specific data structures.
FTPC_EXT.H	Definitions for the Client-level API functions found in FTPC.C.
FCP_EXTR.H	Function prototypes and defined constants for the FTP Client-level command primitives found in FCP.C.
FTP_CFG.H	Configuration definitions for FTP Client and Server. Data structures can be customized by application developer.
NU_FTP.H	This is the only header file required to include in an application to have access to all FTP Client and Server-Level functions.



Nucleus FTP Commands Supported

The following table shows all of the commands that are supported with this version of the Nucleus Extended Protocol Package. If you wish to add more commands, you can easily modify the Control Task routine to add a call for the command, and write your command in `FSP.C` (for server commands) and `FCP.C` (for client commands).

Command	Supported	Meaning
USER	Client/Server	Command that identifies the user.
REIN	Unsupported	This command terminates a USER, flushing all I/O and account information, except to allow any transfer in progress to be completed.
TYPE	Client/Server	The argument specifies the representation type command. Only the Image parameter supported at this time.
STOR	Client/Server	Store command used for storing files.
RNFR	Client/Server	This command specifies the old pathname of the file that is to be renamed. This command must be immediately followed by a "rename to" command specifying the new file pathname.
CWD	Client/Server	This command allows the user to work with a different directory or dataset for file storage or retrieval without altering his login or accounting information.
SYST	Client/Server	System command.
CDUP	Client	Change to Parent Directory.
STOU	Unsupported	Store Unique command.
PASS	Client/Server	Command that specifies the user's password.
QUIT	Client/Server	Logout command.
STRU	Client/Server	File Structure command.
APPE	Client/Server	This command causes the server-DTP to accept the data transferred via the data connection and to store the data in a file at the server site. If the file specified in the pathname exists at the server site, then the data shall be appended to that file; otherwise, the file specified in the pathname shall be created at the server site.

LIST	Client/Server	This command causes a list of directories and filenames to be sent from the server.
STAT	Client/Server	This command shall cause a status response to be sent over the control connection in the form of a reply.
RMD	Client/Server	Remove Directory command.
XRMD	Client/Server	Microsoft command-line Remove Directory command.
ACCT	Client/Server	Account command.
PORT	Client/Server	The argument is a HOST-PORT specification for the data port to be used in data connection.
MODE	Client/Server	Transfer Mode command. Only MODE=stream command is supported.
ALLO	Unsupported	This command may be required by some servers to reserve sufficient storage to accommodate the new file to be transferred.
ABOR	Server	This command tells the server to abort the previous FTP service command and any associated transfer of data.
NLST	Client/Server	This command causes a list of filenames to be sent.
HELP	Client/Server	This command causes the server to send helpful information regarding its implementation status over the control connection to the user.
PWD	Client/Server	Print Current Working Directory command.
SMNT	Unsupported	Structure Mount command.
XMKD	Client/Server	Microsoft command-line FTP Make Directory.
PASV	Client/Server	This command requests the server to "listen" on a data port (which is not its default data port) and to wait for a connection rather than initiate one upon receipt of a transfer command.
RETR	Client/Server	The Retrieve command.
REST	Unsupported	The argument field represents the server marker at which file transfer is to be restarted.
DELE	Client/Server	This command causes the file specified in the pathname to be deleted.

MODE	Client/Server	Transfer Mode command. Only MODE=stream command is supported.
ALLO	Unsupported	This command may be required by some servers to reserve sufficient storage to accommodate the new file to be transferred.
ABOR	Server	This command tells the server to abort the previous FTP service command and any associated transfer of data.
NLST	Client/Server	This command causes a list of filenames to be sent.
HELP	Client/Server	This command causes the server to send helpful information regarding its implementation status over the control connection to the user.
PWD	Client/Server	Print Current Working Directory command.
SMNT	Unsupported	Structure Mount command.
XMKD	Client/Server	Microsoft command-line FTP Make Directory.
PASV	Client/Server	This command requests the server to "listen" on a data port (which is not its default data port) and to wait for a connection rather than initiate one upon receipt of a transfer command.
RETR	Client/Server	The Retrieve command.
REST	Unsupported	The argument field represents the server marker at which file transfer is to be restarted.
DELE	Client/Server	This command causes the file specified in the pathname to be deleted.
SITE	Unsupported	This command provides site parameters.
NOOP	Client/Server	This command does not affect any parameters or previously entered commands.
MKD	Client/Server	Make Directory command.
XPWD	Client/Server	Microsoft command-line Print Current Working Directory Command.

FTP Common Defines and Data Structures

This section will cover data structures and defines that are used with both the Nucleus FTP Server and Client.

Defines

The following are return codes used within the Nucleus FTP Server and Client:

Define Variable	Value
FTP_CMD_RECEIVED	1
FTP_EOF	0
FTP_SUCCESS	0
FTP_INVALID_CLIENT	-1501
FTP_INVALID_TASK	-1502
FTP_STACK_ERROR	-1503
FTP_INVALID_USER	-1504
FTP_INVALID_PASSWORD	-1505
FTP_INVALID_TYPE_CODE	-1506
FTP_BUFFER_OVERRUN	-1507
FTP_BAD_HOST	-1508
FTP_BAD_RESPONSE	-1509
FTP_TIMEOUT	-1510
FTP_NEED_PASSWORD	-1511
FTP_NEED_ACCOUNT	-1512
FTP_INVALID_ACCOUNT	-1513
FTP_INVALID_STRU_CODE	-1514
FTP_INVALID_MODE_CODE	-1515
FTP_INVALID_BUFFER	-1516
FTP_TRANSFER_ABORT	-1517
FTP_FILE_ERROR	-1518
FTP_BAD_MSG_FORMAT	-1519
FTP_REPLY_BUFFER_OVERRUN	-1520
FTP_INVALID_IP_ADDR	-1521
FTP_BAD_CMD_FORMAT	-1522
FTP_BAD_FILE_DESCRIPTOR	-1523
FTP_WRITE_FAILED	-1524
FTP_FILE_NOT_FOUND	-1525
FTP_NO_FILE_DESCRIPTOR_AVAIL	-1526
FTP_FILE_ALREADY_EXISTS	-1527

Define Variable	Value
FTP_SPECIAL_ACCESS_ATTEMPTED	-1528
FTP_INVALID_FILE_POINTER	-1529
FTP_UNKNOWN_FILE_ERROR	-1530
FTP_SYNTAX_ERROR	-1531
FTP_REGISTRATION_FAILURE	-1532
FTP_OPEN_DRIVE_FAILURE	-1533
FTP_CURRENT_DIR_FAILURE	-1534
FTP_MEMORY	-1535
FTP_SERVICE_UNAVAILABLE	-1536
FTP_CMD_NOT_IMPLEMENTED	-1537
FTP_BAD_CMD_SEQUENCE	-1538
FTP_FILE_UNAVAILABLE	-1539
FTP_INVALID_FILE_NAME	-1540
FTP_CMD_UNRECOGNIZED	-1541

The following are internal constants used by the Nucleus FTP Server and Client:

Define Variable	Value
FTP_WELLKNOWN	21
FTP_VALID_PATTERN	0x10229700L
FILENAME_SIZE	12
FTP_TYPE_IMAGE	0
FTP_TYPE_ASCII	1
FTP_TYPE_LOCAL8	0

Data Structures

This section will cover the structures used within the Nucleus FTP Server and Client.

The `IP_ADDR_STRUCT` is used to contain the Client's IP address and the Port Number when a data connection is made.

```
typedef struct IP_ADDR_STRUCT
{
    UCHAR        ip_num[4];
    SHORT        port_num;
} IP_ADDR;
```

FTP Server-Level Defines and Data Structures

This section will cover defines and data structures that are used in the Nucleus FTP Server. This section is broken up into two parts. The first part consists of defines used within the Nucleus FTP Server. The second part contains Structure Definitions used within the Nucleus FTP Server.

Defines

The following are internal constants used by the Nucleus FTP Server:

Define Variable	Value
FTPS_COMMAND_BUFFER_SIZE	8
FTPS_REPLY_BUFFER_SIZE	512
FTPS_GENERIC_BUFF_SIZE	256
FTPS_TRANSFER_BUFF_SIZE	512
FTPS_INACT_TIMEOUT	(300 * SCK_Ticks_Per_Second)
FTPS_DATA_TIMEOUT	(300 * SCK_Ticks_Per_Second)
FTP_GET_CURRENT_YEAR	2001

The following is a list of task-related variables used by the Nucleus FTP Server. These values are optimal, but if you add more functionality, you may need to increase the STACK SIZE values.

Define Variable	Value
CLEANER_QUEUE_SIZE	10
FTPS_SERVER_DEFAULT_PORT	21
FTPS_SERVER_MAX_PENDING	10
FTPS_MASTER_STACK_SIZE	5000
FTPS_CLEANER_STACK_SIZE	2000
FTPS_CONTROL_STACK_SIZE	4000
FTPS_DATA_SEND_STACK_SIZE	2000
FTPS_DATA_RECV_STACK_SIZE	3000
FTPS_DATA_DIR_STACK_SIZE	2000
MAX_FTPS_CONNECTIONS	120 /* max. TCP/IP connects is 120 */
CONTROL_PRIORITY	100
DATA_PRIORITY	CONTROL_PRIORITY + 2
DATA_TIME_SLICE	TICKS_PER_SECOND * 20 /* 20 seconds*/



FTP_GET_CURRENT_YEAR

When the client requests Nucleus FTP Server for a directory listing, FTP Server checks to see whether the current year is the same as the year the file was created. If the checks passes then the server sends the time else it sends the year as part of the directory listing to the client.

This macro returns the current year, The user can define this macro to be your own function that return the current year.

```
#define FTP_GET_CURRENT_YEAR    2001
```

Data Structures

This section will cover the structures used within the Nucleus FTP Server.

The following is the `FLAGS` structure used by the Nucleus FTP Server to keep track of the FTP commands it is processing:

`USER`, `PASS`, `TYPE`, `MODE`, `STRU`, `RNFR`, and `tasking`.

```
typedef struct FLAGS
{
    INT16    userFlag;
    INT16    passFlag;
    INT16    typeFlag;
    INT16    modeFlag;
    INT16    struFlag;
    INT16    rnfrFlag;
    INT16    taskFlag;
    INT16    pasvFlag;
} CMD_FLAGS;
```

The following structure is `FTP_SERVER_STRUCT`. This structure is the main structure used within the Nucleus FTP Server and is updated with every connection made.

```
typedef struct FTP_SERVER_STRUCT
{
    UNSIGNED    validPattern;
    NU_TASK     *taskID;
    CHAR        taskName[8];
    DATA_ELEMENT taskStatus;
    UNSIGNED    scheduleCount;
    OPTION      priority;
    OPTION      preempt;
    UNSIGNED    timeSlice;
    VOID        *stackBase;
    UNSIGNED    stackSize;
    UNSIGNED    minStack;
    struct addr_struct serverAddr;
    struct addr_struct serverDataAddr;
    struct addr_struct clientAddr;
    struct addr_struct clientDataAddr;
```



```

    INT          socketd;
    INT          dataSocketd;
    INT          transferType;
    INT          lastError;
    INT          stackError;
    INT          replyIndex, replyTail;
    INT          defaultDrive;
    CHAR         replyBuff[FTPS_REPLY_BUFFER_SIZE];
    CHAR         fileSpec[FTPS_GENERIC_BUFF_SIZE];
    CHAR         path[FTPS_GENERIC_BUFF_SIZE];
    CHAR         *filename;
    CHAR         renamePath[FTPS_GENERIC_BUFF_SIZE];
    CHAR         *renameFile;
    CHAR         currentWorkingDir[FTPS_GENERIC_BUFF_SIZE];
    struct FLAGS cmdFTP;
    CHAR         user[FTP_MAX_ID_LENGTH];
    NU_EVENT_GROUP FTP_Events;
    STATUS        transferStatus;
} FTP_SERVER;

```

FTP Server Simple User Authentication

Defines

The Macros are defined in `FTP/INC/FTP_CFG.H`

The macro `FTP_USE_ANONYMOUS` can be defined if you want to enable the use of anonymous FTP connections. The password for anonymous users is currently an email address. No checking is implemented other than the form of a standard email address. Anonymous FTP is enabled by default(`NU_TRUE`).

```
#define FTP_USE_ANONYMOUS          NU_TRUE
```

Set the maximum length of a username and password as defined in the following `FTP_Password_List[]`.

```
#define FTP_MAX_ID_LENGTH          32
#define FTP_MAX_PW_LENGTH          32
```



Data Structures

This Data Structure that will hold the password list is

```
typedef struct FTPSacct
{
    CHAR id[FTP_MAX_ID_LENGTH];
    CHAR pw[FTP_MAX_PW_LENGTH];
} FTPSACCT;
```

Configurable Variable

A sample list of passwords for simple user authentication is defined in FTP/SRC/FTP_CFG.C. Edit this list to add users to this server. **Note:** No dynamic user administration functions are available, but can be added by the application developer.

```
FTPSACCT FTP_Password_List[] =
{
    { "jon", "doe"},
    { "fred", "fish"},
    { "joe", "blow"},
    { '\0', '\0' }
};
```

FTP Server Level Functions

This section will give a description of all function prototypes used within the FTP Server.

NU_FTP_Server_Init

```
INT32 NU_FTP_Server_Init(VOID)
```

Description

This function is the application layer function that is called to initialize and start the Nucleus FTP Server.

Parameters

None



FTP Client Level Defines and Data Structures

Defines

The FTP Client API uses a number of constants in the form of `#defines`. The majority of these are return codes. Following is a list of defines, their values, and in what category they belong (whether they are a return code, internally used values, or parameter values).

Name	Value
FTPC_MAX_RECV_PACKET	1500
FTPC_MODE_STREAM	0
FTPC_STRU_FILE	0
FTPC_REPLY_BUFFER_SIZE	512
FTPC_GENERIC_BUFF_SIZE	256
FTPC_TRANSFER_BUFF_SIZE	1024
FTPC_INACT_TIMEOUT	(300 * SCK_Ticks_Per_Second)
FTPC_CCONN_TIMEOUT	(30 * SCK_Ticks_Per_Second)
FTPC_DATACT_TIMEOUT	(300 * SCK_Ticks_Per_Second)

The timeout value, `FTPC_INACT_TIMEOUT`, is number of seconds to wait for expected response from the server, on either the control connection or the data connection, before closing the session. This value can be changed to suit individual systems.

The timeout value, `FTPC_CCONN_TIMEOUT`, is number of seconds to wait after attempting to close the control connection before terminating the connection and reporting an error. This value can be changed to suit individual systems.

The timeout value, `FTPC_DATACT_TIMEOUT`, is number of seconds to wait after attempting to send or receive data before the connection is closed and an error is reported. This value can be changed to suit individual systems.

Type Definitions

The FTP Client API relies on a number of structures in order to maintain reentrancy. These structures hold connection-specific data and are required for every call to a function in the API.

FTP_CLIENT

The `FTP_CLIENT` type is used to define the parameters of a specific FTP Client connection. All calls to the API require this parameter. The `FTP_Client_Open` function initializes the fields in this structure and creates the TCP/IP connection. The initialization includes placing a valid pattern in the `valid_pattern` field, which is checked for by every other function in the API, and storing the task ID of the calling task in the `task_id` field. Every open connection is associated with the task that calls `FTP_Client_Open` and may not be used by any other task. Once a connection has



been established, the socket descriptor for the connection is stored in the `socketd` field. The structure contains fields for storage of the host IP address, as well as both data connection IP addresses. Also included is the reply buffer used to store and construct reply messages from the server. Finally, there are fields for storage of a transfer type specifier, the last error type returned by a call to the API, and the error code returned by the stack in case of an `FTP_STACK_ERROR`.

```
typedef struct FTP_CLIENT_STRUCT
{
    UNSIGNED    valid_pattern;
    UNSIGNED    task_id;
    INT         socketd;
    IP_ADDR     *host_addr,
               *local_data_addr,
               *host_data_addr;
    INT         transfer_type;
    INT         last_error;
    INT         stack_error;
    CHAR        reply_buff[FTP_REPLY_BUFFER_SIZE];
    INT         reply_idx,
               reply_tail;
} FTP_CLIENT;
```

FTP Client Level Functions

NU_FTPC_Client_Open

```
INT FTPC_Client_Open(FTP_CLIENT *client, IP_ADDR *host_ip)
```

This function initializes the `FTP_CLIENT` instance `client` and attempts to create a connection to the host defined by `host_ip`. Note that if the `port_num` field of `host_ip` is zero, then the default port for FTP connections will be used, as defined by `FTP_WELLKNOWN`, which defaults to 21. Once a connection has been successfully made through the use of this call, only the task that made the call may use the connection.



NOTE: Do not call this function with a structure that still refers to an open connection, as the old connection will be lost, and the structure overwritten with the data for the new connection.

Parameters

Parameter	Meaning
<code>Client</code>	(<code>FTP_CLIENT *</code>) Pointer to the <code>FTP_Client</code> structure.
<code>host_ip</code>	(<code>IP_ADDR *</code>) Pointer to the ip address of the remote host.

Return Value

Upon successful completion of this service, it returns `FTP_SUCCESS`. Otherwise, it returns `FTP_STACK_ERROR` or `FTP_BAD_HOST`.

NU_FTPC_Client_Close

```
INT FTPC_Client_Close(FTP_CLIENT *client)
```

This function closes an FTP connection between the client and the target host. If an error is returned by this function, the connection is still closed. The error indicates that some internal function failed in its task.

Parameters

Parameter	Meaning
Client	(FTP_CLIENT *) Pointer to the FTP_Client structure.

Return Value

Upon successful completion of this service, it returns FTP_SUCCESS. Otherwise, it returns the following:

```
FTP_INVALID_CLIENT,
FTP_INVALID_TASK,
FTP_STACK_ERROR,
FTP_CMD_UNRECOGNIZED or
FTP_BAD_RESPONSE.
```

NU_FTPC_Client_Login

```
INT FTPC_Client_Login(FTP_CLIENT *client,
                     CHAR *username,
                     CHAR *password)
```

This function attempts to open the account of a particular user, defined by `username`, on the currently connected host. The `password` parameter may be left `NULL` if the host does not require a password. If the parameter is left `NULL` and the host does ask for a password, the function will return `FTP_INVALID_PASSWORD`. If a password is not required by the host, but is provided by the user, it will be ignored. If `username` is `NULL`, the function will attempt to connect as ‘anonymous,’ with no password.

Parameters

Parameter	Meaning
<code>client</code>	(<code>FTP_CLIENT *</code>) Pointer to the <code>FTP_Client</code> structure.
<code>username</code>	(<code>CHAR *</code>) Pointer to a buffer containing the user name.
<code>password</code>	(<code>CHAR *</code>) Pointer to a buffer containing the password.

Return Value

Upon successful completion of this service, it returns `FTP_SUCCESS`. Otherwise, it returns the following:

```
FTP_INVALID_CLIENT,
FTP_INVALID_TASK,
FTP_NEED_ACCOUNT,
FTP_SERVICE_UNAVAILABLE,
FTP_CMD_UNRECOGNIZED,
FTP_BAD_CMD_FORMAT,
FTP_CMD_NOT_IMPLEMENTED,
FTP_BAD_CMD_SEQUENCE,
FTP_INVALID_USER or
FTP_BAD_RESPONSE.
```



NU_FTPC_Client_Get

```
INT FTPC_Client_Get(FTP_CLIENT *client,
                   CHAR *rpath,
                   CHAR *lpath)
```

This function retrieves a file from the host computer specified in `rpath` and stores the file in the local location specified in `lpath`.

Parameters

Parameter	Meaning
<code>client</code>	(FTP_CLIENT *) Pointer to the FTP_Client structure.
<code>rpath</code>	(CHAR *) Pointer to a buffer containing the path of the remote file.
<code>lpath</code>	(CHAR *) Pointer to a buffer containing the path of the local file.

Return Value

Upon successful completion of this service, it returns `FTP_SUCCESS`. Otherwise, it returns the following:

```
FTP_INVALID_CLIENT,
FTP_INVALID_TASK,
FTP_SERVICE_UNAVAILABLE,
FTP_CMD_UNRECOGNIZED,
FTP_BAD_CMD_FORMAT,
FTP_INVALID_USER,
FTP_BAD_RESPONSE,
FTP_FILE_UNAVAILABLE,
FTP_FILE_NOT_FOUND,
FTP_TIMEOUT,
FTP_STACK_ERROR,
FTP_MEMORY or
FTP_FILE_ERROR.
```

NU_FTPC_Client_Put

```
INT FTPC_Client_Put(FTP_CLIENT *client,  
                    CHAR *rpath,  
                    CHAR *lpath)
```

This function transfers a local file specified by `lpath` to the remote host, storing the file at the location specified by `rpath`. If the file exists on the host, and the host is configured to not allow overwrites, an error will be returned.

Parameters

Parameter	Meaning
<code>client</code>	(FTP_CLIENT *) Pointer to the <code>FTP_Client</code> structure.
<code>rpath</code>	(CHAR *) Pointer to a buffer containing the path of the remote file.
<code>lpath</code>	(CHAR *) Pointer to a buffer containing the path of the local file.

Return Value

Upon successful completion of this service, it returns `FTP_SUCCESS`. Otherwise, it returns the following:

```
FTP_INVALID_CLIENT,  
FTP_INVALID_TASK,  
FTP_SERVICE_UNAVAILABLE,  
FTP_CMD_UNRECOGNIZED,  
FTP_BAD_CMD_FORMAT,  
FTP_INVALID_USER,  
FTP_BAD_RESPONSE,  
FTP_FILE_UNAVAILABLEFTP_TIMEOUT,  
FTP_STACK_ERROR,  
FTP_NEED_ACCOUNT,  
FTP_INVALID_FILE_NAME,  
FTP_MEMORY or  
FTP_FILE_ERROR.
```



NU_FTPC_Client_Append_To_File

```
INT FTPC_Client_Append_To_File(FTP_CLIENT *client,
                               CHAR *rpath,
                               CHAR *lpath)
```

This function appends a local file specified by `lpath` to the remote file at the location specified by `rpath`.

Parameters

Parameter	Meaning
<code>client</code>	(FTP_CLIENT *) Pointer to the FTP_Client structure.
<code>rpath</code>	(CHAR *) Pointer to a buffer containing the path of the remote file.
<code>lpath</code>	(CHAR *) Pointer to a buffer containing the path of the local file.

Return Value

Upon successful completion of this service, it returns `FTP_SUCCESS`. Otherwise, it returns the following:

```
FTP_INVALID_CLIENT,
FTP_INVALID_TASK,
FTP_SERVICE_UNAVAILABLE,
FTP_CMD_UNRECOGNIZED,
FTP_BAD_CMD_FORMAT,
FTP_INVALID_USER,
FTP_BAD_RESPONSE,
FTP_TIMEOUT,
FTP_STACK_ERROR,
FTP_MEMORY or
FTP_FILE_ERROR.
```

NU_FTPC_Client_ChDir

```
INT FTPC_Client_ChDir(FTP_CLIENT *client, CHAR *path)
```

FTPC_Client_ChDir attempts to change the currently active directory on the remote system.

Parameters

Parameter	Meaning
client	(FTP_CLIENT *) Pointer to the FTP_Client structure.
path	(CHAR *) Pointer to a buffer containing the remote path.

Return Value

Upon successful completion of this service, it returns FTP_SUCCESS. Otherwise, it returns the following:

FTP_INVALID_CLIENT,
FTP_INVALID_TASK,
FTP_SERVICE_UNAVAILABLE,
FTP_STACK_ERROR,
FTP_INVALID_USER,
FTP_FILE_NOT_FOUND,
FTP_BAD_RESPONSE,
FTP_CMD_UNRECOGNIZED
FTP_BAD_CMD_FORMAT or
FTP_CMD_NOT_IMPLEMENTED.



NU_FTPC_Client_Dir

```
INT FTPC_Client_Dir(FTP_CLIENT *client,
                    CHAR *buffer,
                    INT buff_size,
                    CHAR *filespec)
```

This function performs a directory listing of the current active remote directory, including only those files that match the specification stored in `filespec`. The listing is stored in `buffer`. If the listing exceeds the amount of space available in `buffer` (as defined by `buff_size`), the function discards the excess input and returns the `FTP_BUFFER_OVERRUN` error code. The user can then choose to use the incomplete data or call the function again with a larger buffer.

Parameters

Parameter	Meaning
<code>client</code>	(<code>FTP_CLIENT *</code>) Pointer to the <code>FTP_Client</code> structure.
<code>buffer</code>	(<code>CHAR *</code>) Pointer to a buffer that will hold the directory information.
<code>buff_size</code>	(<code>INT</code>) Size of the buffer.
<code>filespec</code>	(<code>CHAR *</code>) Pointer to a buffer containing the specification of the files to be listed.

Return Value

Upon successful completion of this service, it returns `FTP_SUCCESS`. Otherwise, it returns the following:

```
FTP_INVALID_CLIENT,
FTP_INVALID_TASK,
FTP_SERVICE_UNAVAILABLE,
FTP_CMD_UNRECOGNIZED,
FTP_BAD_CMD_FORMAT,
FTP_INVALID_USER,
FTP_BAD_RESPONSE,
FTP_TIMEOUT,
FTP_STACK_ERROR or
FTP_INVALID_BUFFER.
```



NU_FTPC_Client_Nlist

```
INT FTPC_Client_Nlist(FTP_CLIENT *client,  
                      CHAR *buffer,  
                      INT buff_size,  
                      CHAR *filespec)
```

This function performs a directory listing of the remote host, including only those files that match the specification stored in `filespec`. The listing is stored in `buffer`. If the listing exceeds the amount of space available in `buffer` (as defined by `buff_size`) the listing will be truncated.

Parameters

Parameter	Meaning
<code>client</code>	(<code>FTP_CLIENT *</code>) Pointer to the <code>FTP_Client</code> structure.
<code>buffer</code>	(<code>CHAR *</code>) Pointer to a buffer that will hold the directory information.
<code>buff_size</code>	(<code>INT</code>) Size of the buffer.
<code>filespec</code>	(<code>CHAR *</code>) Pointer to a buffer containing the specification of the files to be listed.

Return Value

Upon successful completion of this service, it returns `FTP_SUCCESS`. Otherwise, it returns the following:

```
FTP_INVALID_CLIENT,  
FTP_INVALID_TASK,  
FTP_SERVICE_UNAVAILABLE,  
FTP_CMD_UNRECOGNIZED,  
FTP_BAD_CMD_FORMAT,  
FTP_INVALID_USER,  
FTP_BAD_RESPONSE,  
FTP_TIMEOUT,  
FTP_STACK_ERROR or  
FTP_INVALID_BUFFER.
```



NU_FTPC_Client-Tran_Type

```
INT FTPC_Client-Tran_Type(FTP_CLIENT *client, INT type)
```

This function alters the defined data transfer type. The three possible codes are: FTP_TYPE_ASCII, FTP_TYPE_IMAGE, and FTP_TYPE_LOCAL8 (which is functionally identical to IMAGE).

Parameters

Parameter	Meaning
client	(FTP_CLIENT *) Pointer to the FTP_Client structure.
type	(INT) The data transfer type.

Return Value

Upon successful completion of this service, it returns FTP_SUCCESS. Otherwise, it returns the following:

```
FTP_INVALID_CLIENT,
FTP_INVALID_TASK,
FTP_STACK_ERROR,
FTP_INVALID_TYPE_CODE,
FTP_SERVICE_UNAVAILABLE,
FTP_CMD_UNRECOGNIZED,
FTP_BAD_CMD_FORMAT,
FTP_CMD_NOT_IMPLEMENTED,
FTP_INVALID_USER or
FTP_BAD_RESPONSE.
```

NU_FTPC_Client_RmDir

```
INT FTPC_Client_RmDir(FTP_CLIENT *client, CHAR *path)
```

This function removes the directory specified by `path` from the server. An error is returned if the directory does not exist, the directory cannot be removed (in use, or not empty), or if the user does not have the appropriate privilege level.

Parameters

Parameter	Meaning
<code>client</code>	(FTP_CLIENT *) Pointer to the FTP_Client structure.
<code>path</code>	(CHAR *) Pointer to the character string containing the path of the target directory.

Return Value

Upon successful completion of this service, it returns `FTP_SUCCESS`. Otherwise, it returns the following:

```
FTP_INVALID_CLIENT,  
FTP_INVALID_TASK,  
FTP_STACK_ERROR,  
FTP_SERVICE_UNAVAILABLE,  
FTP_CMD_UNRECOGNIZED,  
FTP_BAD_CMD_FORMAT,  
FTP_CMD_NOT_IMPLEMENTED,  
FTP_BAD_CMD_SEQUENCE,  
FTP_INVALID_USER,  
FTP_FILE_UNAVAILABLE, or  
FTP_BAD_RESPONSE.
```



NU_FTPC_Client_MkDir

```
INT FTPC_Client_MkDir(FTP_CLIENT *client, CHAR *path)
```

This function creates a directory with the name stored in `path`. An error is returned if `path` refers to a directory that already exists, the path of the new directory is invalid (an absolute path where one of the parent directories does not exist), or if the user does not have the appropriate privilege level.

Parameters

Parameter	Meaning
<code>client</code>	(FTP_CLIENT *) Pointer to the FTP_Client structure.
<code>path</code>	(CHAR *) Pointer to the character string containing the path of the new directory.

Return Value

Upon successful completion of this service, it returns `FTP_SUCCESS`. Otherwise, it returns the following:

```
FTP_INVALID_CLIENT,
FTP_INVALID_TASK,
FTP_STACK_ERROR,
FTP_SERVICE_UNAVAILABLE
FTP_CMD_UNRECOGNIZED,
FTP_BAD_CMD_FORMAT,
FTP_CMD_NOT_IMPLEMENTED,
FTP_INVALID_USER,
FTP_FILE_UNAVAILABLE or
FTP_BAD_RESPONSE.
```

NU_FTPC_Client_Status

```
INT FTPC_Client_Status(FTP_CLIENT *client,  
                       CHAR *path,  
                       CHAR *buffer,  
                       INT bufsize)
```

This function returns the status of the entity defined by path. The status information is returned in `buffer`, with a maximum length of `bufsize`. An error is returned if the entity in `path` does not exist.

Parameters

Parameter	Meaning
<code>client</code>	(<code>FTP_CLIENT *</code>) Pointer to the <code>FTP_Client</code> structure.
<code>path</code>	(<code>CHAR *</code>) Pointer to the character string containing the object that will be checked.
<code>buffer</code>	(<code>CHAR *</code>) Pointer to a buffer to hold the status information.
<code>bufsize</code>	(<code>INT</code>) Size of the buffer.

Return Value

Upon successful completion of this service, it returns `FTP_SUCCESS`. Otherwise, it returns the following:

```
FTP_INVALID_CLIENT,  
FTP_INVALID_TASK,  
FTP_STACK_ERROR,  
FTP_SERVICE_UNAVAILABLE,  
FTP_CMD_UNRECOGNIZED,  
FTP_BAD_CMD_FORMAT,  
FTP_CMD_NOT_IMPLEMENTED,  
FTP_FILE_UNAVAILABLE or  
FTP_BAD_RESPONSE.
```



NU_FTPC_Client_Rename_File

```
INT FTPC_Client_Rename_File(FTP_CLIENT *client,
                             CHAR *old_file_name,
                             CHAR *new_file_name)
```

This function changes the name of a remote file from `old_file_name` to `new_file_name`. An error is returned if the file does not exist or if a file already exists with the name in `new_file_name`.

Parameters

Parameter	Meaning
<code>client</code>	(FTP_CLIENT *) Pointer to the FTP_Client structure.
<code>old_file_name</code>	(CHAR *) Pointer to the old file name.
<code>new_file_name</code>	(CHAR *) Pointer to the new file name.

Return Value

Upon successful completion of this service, it returns `FTP_SUCCESS`. Otherwise, it returns the following:

```
FTP_INVALID_CLIENT,
FTP_INVALID_TASK,
FTP_STACK_ERROR,
FTP_SERVICE_UNAVAILABLE,
FTP_CMD_UNRECOGNIZED,
FTP_BAD_CMD_FORMAT,
FTP_CMD_NOT_IMPLEMENTED,
FTP_BAD_CMD_SEQUENCE,
FTP_INVALID_USER,
FTP_NEED_ACCOUNT,
FTP_INVALID_FILE_NAME,
FTP_BAD_RESPONSE or
FTP_FILE_UNAVAILABLE.
```

NU_FTPC_Client_Tran_Mode

```
INT FTPC_Client_Tran_Mode(FTP_CLIENT *client, INT mode)
```

This function alters the defined data transfer mode. The only defined mode is `STREAM`. The constant `FTP_MODE_STREAM` defines the value that this function uses to represent this mode.

Parameters

Parameter	Meaning
<code>client</code>	(<code>FTP_CLIENT *</code>) Pointer to the <code>FTP_Client</code> structure.
<code>mode</code>	(<code>INT</code>) Transfer mode.

Return Value

Upon successful completion of this service, it returns `FTP_SUCCESS`. Otherwise, it returns the following:

```
FTP_INVALID_CLIENT,  
FTP_INVALID_TASK,  
FTP_INVALID_MODE_CODE  
FTP_STACK_ERROR,  
FTP_SERVICE_UNAVAILABLE,  
FTP_CMD_UNRECOGNIZED,  
FTP_BAD_CMD_FORMAT,  
FTP_CMD_NOT_IMPLEMENTED,  
FTP_INVALID_USER or  
FTP_BAD_RESPONSE.
```



FTP Client-Level Command Primitives

NU_FCP_Reply_Read

```
INT FCP_Reply_Read(FTP_CLIENT *client,
                  unsigned char *buffer,
                  INT bufsize,
                  INT timeout)
```

This general-purpose function retrieves a single valid FTP reply message from the command connection and returns the message to the calling function. Each of the FTP command primitives uses this function to receive the reply from the server.

The function returns a single FTP reply message in the memory space pointed to by `buffer`, returning no more than `bufsize` characters. The function will wait for `timeout` timer ticks before returning an error if no data becomes available during that time.

Parameters

Parameter	Meaning
<code>client</code>	(<code>FTP_CLIENT *</code>) Pointer to the <code>FTP_CLIENT</code> structure.
<code>buffer</code>	(<code>unsigned char *</code>) Pointer to the destination buffer.
<code>bufsize</code>	(<code>INT</code>) The size of the destination buffer.
<code>timeout</code>	(<code>INT</code>) The timeout value for the read.

Return Value

Upon successful completion of this service, it returns `NU_SUCCESS`. Otherwise, it returns the following:

```
FTP_TIMEOUT,
FTP_STACK_ERROR,
FTP_BAD_MSG_FORMAT or
FTP_REPLY_BUFFER_OVERRUN.
```

NU_FCP_Client_Verify_Caller

```
INT FCP_Client_Verify_Caller(FTP_CLIENT *client)
```

This general-purpose function verifies that the currently running task is the ‘owner’ of the FTP session defined by `client`.

Parameters

Parameter	Meaning
<code>client</code>	(FTP_CLIENT *) Pointer to the FTP_CLIENT structure.

Return Value

Upon successful completion of this service, it returns `NU_SUCCESS`. Otherwise, it returns `FTP_INVALID_CLIENT` or `FTP_INVALID_TASK`.



NU_FCP_Client_Tran_Ack

```
INT FCP_Client_Tran_Ack(FTP_CLIENT *client,
                        INT ignore_flag)
```

This general-purpose function performs the final cleanup following any FTP command that required a data connection. The `ignore_flag` parameter is set if an error condition occurred during the transfer at the client side. This allows the function to be used to clean the final reply message.

Parameters

Parameter	Meaning
<code>client</code>	(FTP_CLIENT *) Pointer to the FTP_Client structure.
<code>ignore_flag</code>	(INT) Flag to determine whether or not to use the incoming message.

Return Value

Upon successful completion of this service, it returns `NU_SUCCESS`. Otherwise, it returns the following:

```
FTP_STACK_ERROR,
FTP_SERVICE_UNAVAILABLE,
FTP_TRANSFER_ABORT,
FTP_FILE_UNAVAILABLE,
FTP_TRANSFER_ABORT,
FTP_SYNTAX_ERROR,
FTP_FILE_NOT_FOUND or
FTP_BAD_RESPONSE.
```



NU_FCP_Client_TYPE

```
INT FCP_Client_TYPE(FTP_CLIENT *client, INT type)
```

This function provides a primitive to send an FTP TYPE message to the server. Supported types include ASCII, IMAGE (binary), and LOCAL 8 (identical to IMAGE, but necessary for RFC compliance).

Parameters

Parameter	Meaning
client	(FTP_CLIENT *) Pointer to the FTP_Client structure.
type	(INT) Code for transfer type.

Return Value

Upon successful completion of this service, it returns NU_SUCCESS. Otherwise, it returns the following:

```
FTP_STACK_ERROR,  
FTP_INVALID_TYPE_CODE,  
FTP_SERVICE_UNAVAILABLE,  
FTP_CMD_UNRECOGNIZED,  
FTP_BAD_CMD_FORMAT,  
FTP_CMD_NOT_IMPLEMENTED,  
FTP_INVALID_USER or  
FTP_BAD_RESPONSE.
```



NU_FCP_Client_MODE

```
INT FCP_Client_MODE(FTP_CLIENT *client, INT mode)
```

This function provides a primitive to send an FTP MODE message to the server. The sole supported mode is `STREAM`.

Parameters

Parameter	Meaning
<code>client</code>	(<code>FTP_CLIENT *</code>) Pointer to the <code>FTP_Client</code> structure.
<code>mode</code>	(<code>INT</code>) Code for transfer mode.

Return Value

Upon successful completion of this service, it returns `NU_SUCCESS`. Otherwise, it returns the following:

```
FTP_STACK_ERROR,
FTP_SERVICE_UNAVAILABLE,
FTP_CMD_UNRECOGNIZED,
FTP_BAD_CMD_FORMAT,
FTP_CMD_NOT_IMPLEMENTED,
FTP_INVALID_USER or
FTP_BAD_RESPONSE.
```

NU_FCP_Client_STRU

```
INT FCP_Client_STRU(FTP_CLIENT *client, INT structure)
```

This function provides a primitive to send an FTP STRU message to the server. The sole supported structure type is `FILE`.

Parameters

Parameter	Meaning
<code>client</code>	(<code>FTP_CLIENT *</code>) Pointer to the <code>FTP_Client</code> structure.
<code>structure</code>	(<code>INT</code>) Code for transfer structure type.

Return Value

Upon successful completion of this service, it returns `NU_SUCCESS`. Otherwise, it returns the following:

```
FTP_INVALID_STRU_CODE,  
FTP_STACK_ERROR,  
FTP_SERVICE_UNAVAILABLE,  
FTP_CMD_UNRECOGNIZED,  
FTP_BAD_CMD_FORMAT,  
FTP_CMD_NOT_IMPLEMENTED,  
FTP_INVALID_USER or  
FTP_BAD_RESPONSE.
```



NU_FCP_Client_PORT

```
INT FCP_Client_PORT(FTP_CLIENT *client,
                    struct addr_struct *servaddr)
```

This function provides a primitive to send an FTP PORT message to the server.

Parameters

Parameter	Meaning
client	(FTP_CLIENT *) Pointer to the FTP_Client structure.
servaddr	(STRUCT_ADDR_STRUCT*) Pointer to the structure containing the address to be sent to the server.

Return Value

Upon successful completion of this service, it returns NU_SUCCESS. Otherwise, it returns the following:

```
FTP_STACK_ERROR,
FTP_SERVICE_UNAVAILABLE,
FTP_CMD_UNRECOGNIZED,
FTP_BAD_CMD_FORMAT,
FTP_INVALID_USER or
FTP_BAD_RESPONSE.
```

NU_FCP_Client_LIST

```
INT FCP_Client_LIST(FTP_CLIENT *client, CHAR *filespec)
```

This function provides a primitive to send an FTP LIST message to the server.

Parameters

Parameter	Meaning
client	(FTP_CLIENT *) Pointer to the FTP_Client structure.
filespec	(CHAR *) Pointer to the string containing the directory to be retrieved.

Return Value

Upon successful completion of this service, it returns NU_SUCCESS. Otherwise, it returns the following:

FTP_TIMEOUT,
FTP_STACK_ERROR or
FTP_INVALID_BUFFER.



NU_FCP_Client_RETR

```
INT FCP_Client_RETR(FTP_CLIENT *client, CHAR *filespec)
```

This function provides a primitive to send an FTP RETR message to the server. If a path is included in the string, it refers to the path of the source file, not the destination file. Wildcards are not allowed.

Parameters

Parameter	Meaning
client	(FTP_CLIENT *) Pointer to the FTP_Client structure.
filespec	(CHAR *) Pointer to the string containing the name of the file to be retrieved.

Return Value

Upon successful completion of this service, it returns NU_SUCCESS. Otherwise, it returns the following:

```
FTP_STACK_ERROR,
FTP_SERVICE_UNAVAILABLE,
FTP_FILE_UNAVAILABLE,
FTP_CMD_UNRECOGNIZED,
FTP_BAD_CMD_FORMAT,
FTP_INVALID_USER,
FTP_FILE_NOT_FOUND or
FTP_BAD_RESPONSE.
```

NU_FCP_Client_STOR

```
INT FCP_Client_STOR(FTP_CLIENT *client, CHAR *filespec)
```

This function provides a primitive to send an FTP STOR message to the server. If a path is included in the string, it refers to the path of the destination file rather than the path of the local file. Wildcards are not allowed.

Parameters

Parameter	Meaning
client	(FTP_CLIENT *) Pointer to the FTP_Client structure.
filespec	(CHAR *) Pointer to the string containing the name of the file to be sent.

Return Value

Upon successful completion of this service, it returns NU_SUCCESS. Otherwise, it returns the following:

```
FTP_STACK_ERROR,  
FTP_SERVICE_UNAVAILABLE,  
FTP_FILE_UNAVAILABLE,  
FTP_MEMORY,  
FTP_CMD_UNRECOGNIZED,  
FTP_BAD_CMD_FORMAT,  
FTP_INVALID_USER,  
FTP_NEED_ACCOUNT,  
FTP_INVALID_FILE_NAME or  
FTP_BAD_RESPONSE.
```



NU_FCP_Client_APPE

```
INT FCP_Client_APPE(FTP_CLIENT *client, CHAR *filespec)
```

This function provides a primitive to send an FTP APPE message to the server. Any data sent over the data connection will be added to a file of that name on the host system. If a path is included in the string, it refers to the file on the host rather than the file to be sent. Wildcards are not allowed.

Parameters

Parameter	Meaning
client	(FTP_CLIENT *) Pointer to the FTP_Client structure.
filespec	(CHAR *) Pointer to the string containing the name of the file to be appended.

Return Value

Upon successful completion of this service, it returns NU_SUCCESS. Otherwise, it returns the following:

```
FTP_STACK_ERROR,
FTP_SERVICE_UNAVAILABLE,
FTP_FILE_UNAVAILABLE,
FTP_MEMORY,
FTP_CMD_UNRECOGNIZED,
FTP_BAD_CMD_FORMAT,
FTP_CMD_NOT_IMPLEMENTED,
FTP_INVALID_USER,
FTP_NEED_ACCOUNT,
FTP_FILE_UNAVAILABLE,
FTP_INVALID_FILE_NAME or
FTP_BAD_RESPONSE.
```

NU_FCP_Client_NLST

```
INT FCP_Client_NLST(FTP_CLIENT *client, CHAR *filespec)
```

This function provides a primitive to send an FTP NLST message to the server.

Parameters

Parameter	Meaning
client	(FTP_CLIENT *) Pointer to the FTP_Client structure.
filespec	(CHAR *) Pointer to the string containing the directory to be retrieved.

Return Value

Upon successful completion of this service, it returns NU_SUCCESS. Otherwise, it returns the following:

```
FTP_STACK_ERROR,  
FTP_SERVICE_UNAVAILABLE,  
FTP_FILE_UNAVAILABLE,  
FTP_CMD_UNRECOGNIZED,  
FTP_BAD_CMD_FORMAT,  
FTP_CMD_NOT_IMPLEMENTED,  
FTP_INVALID_USER or  
FTP_BAD_RESPONSE.
```



NU_FCP_Client_USER

```
INT FCP_Client_USER(FTP_CLIENT *client, CHAR *username)
```

This function provides a primitive to send an FTP USER message to the server. This function has multiple 'success' codes, including the generic success (requiring no further action), and the 'need password' success, where the command was received and processed properly but does not have enough information to establish a connection.

Parameters

Parameter	Meaning
client	(FTP_CLIENT *) Pointer to the FTP_Client structure.
username	(CHAR *) Pointer to the string containing the username.

Return Value

Upon successful completion of this service, it returns NU_SUCCESS. Otherwise, it returns the following:

```
FTP_STACK_ERROR,
FTP_NEED_PASSWORD,
FTP_NEED_ACCOUNT,
FTP_SERVICE_UNAVAILABLE,
FTP_CMD_UNRECOGNIZED,
FTP_BAD_CMD_FORMAT,
FTP_INVALID_USER or
FTP_BAD_RESPONSE.
```

NU_FCP_Client_PASS

```
INT FCP_Client_PASS(FTP_CLIENT *client, CHAR *password)
```

This function provides a primitive to send an FTP PASS message to the server. This function has multiple success conditions, including the generic 'success' message, indicating that no further action is necessary, and the 'need account' message, indicating that an account name (transferred via the ACCT function) is necessary. Note that this function will (most likely) return an error if it is not preceded by a USER command. The error is returned based on the implementation of the server.

Parameters

Parameter	Meaning
client	(FTP_CLIENT *) Pointer to the FTP_Client structure.
password	(CHAR *) Pointer to the string containing the password.

Return Value

Upon successful completion of this service, it returns NU_SUCCESS. Otherwise, it returns the following:

```
FTP_STACK_ERROR,  
FTP_CMD_NOT_IMPLEMENTED,  
FTP_SUCCESS,  
FTP_NEED_ACCOUNT,  
FTP_SERVICE_UNAVAILABLE,  
FTP_CMD_UNRECOGNIZED,  
FTP_BAD_CMD_FORMAT,  
FTP_BAD_CMD_SEQUENCE,  
FTP_INVALID_USER or  
FTP_BAD_RESPONSE.
```

NU_FCP_Client_ACCT

```
INT FCP_Client_ACCT(FTP_CLIENT *client, CHAR *account)
```

This function provides a primitive to send an FTP ACCT message to a server. This function will (most likely) return an error, if it is not directly preceded by either a USER or a PASS command. The error is returned based on the implementation of the server.

Parameters

Parameter	Meaning
client	(FTP_CLIENT *) Pointer to the FTP_Client structure.
account	(CHAR *) Pointer to the string containing the account name.

Return Value

Upon successful completion of this service, it returns NU_SUCCESS. Otherwise, it returns the following:

```
FTP_STACK_ERROR,
FTP_INVALID_ACCOUNT,
FTP_CMD_NOT_IMPLEMENTED,
FTP_SERVICE_UNAVAILABLE,
FTP_CMD_UNRECOGNIZED,
FTP_BAD_CMD_FORMAT,
FTP_BAD_CMD_SEQUENCE,
FTP_INVALID_USER or
FTP_BAD_RESPONSE.
```

NU_FCP_Client_CWD

NU_FCP_Client_XCWD

```
INT FCP_Client_CWD(FTP_CLIENT *client, CHAR *path)
INT FCP_Client_XCWD(FTP_CLIENT *client, CHAR *path)
```

This function provides a primitive to send an FTP CWD (change working directory) message to the server. An error from the server indicates that the path was either not valid or was inaccessible (privileged access areas, etc.). Note that the XCWD version is for backward compatibility with older servers that may not recognize the newer forms. The CWD variant will call the XCWD variant if an unrecognized command error is received. The XCWD function need not ever be used, and is here for reference only.

Parameters

Parameter	Meaning
client	(FTP_CLIENT *) Pointer to the FTP_Client structure.
path	(CHAR *) Pointer to the string containing the new directory.

Return Value

Upon successful completion of this service, it returns NU_SUCCESS. Otherwise, it returns the following:

```
FTP_STACK_ERROR,
FTP_SERVICE_UNAVAILABLE,
FTP_CMD_UNRECOGNIZED
FTP_BAD_CMD_FORMAT,
FTP_CMD_NOT_IMPLEMENTED,
FTP_INVALID_USER,
FTP_FILE_NOT_FOUND or
FTP_BAD_RESPONSE.
```



NU_FCP_Client_MKD

NU_FCP_Client_XMKD

```
INT FCP_Client_MKD(FTP_CLIENT *client, CHAR *path)
INT FCP_Client_XMKD(FTP_CLIENT *client, CHAR *path)
```

This function provides a primitive to send an FTP MKD (make directory) message to the server. An error from the server indicates that the path was invalid or that creating a directory was a privileged action. Note that the XMKD version is for backward compatibility with older servers that may not recognize the newer forms. The MKD variant will call the XMKD variant if an unrecognized command error is received. The XMKD function need not ever be used, and is here for reference only.

Parameters

Parameter	Meaning
client	(FTP_CLIENT *) Pointer to the FTP_Client structure.
path	(CHAR *) Pointer to the string containing the new directory.

Return Value

Upon successful completion of this service, it returns NU_SUCCESS. Otherwise, it returns the following:

```
FTP_STACK_ERROR,
FTP_SERVICE_UNAVAILABLE,
FTP_CMD_UNRECOGNIZED,
FTP_BAD_CMD_FORMAT,
FTP_CMD_NOT_IMPLEMENTED,
FTP_INVALID_USER,
FTP_FILE_UNAVAILABLE or
FTP_BAD_RESPONSE.
```

NU_FCP_Client_RMD

NU_FCP_Client_XRMD

```
INT FCP_Client_RMD(FTP_CLIENT *client, CHAR *path)
INT FCP_Client_XRMD(FTP_CLIENT *client, CHAR *path)
```

This function provides a primitive to send an FTP RMD (remove directory) message to a server. An error from the server indicates that the path was invalid, undeletable (under MSDOS, still had files in it), or that removing a directory was a privileged action. Note that the XRMD version is for backward compatibility with older servers that may not recognize the newer forms. The RMD variant will call the XRMD variant if an unrecognized command error is received. The XRMD function need not ever be used and is here for reference only.

Parameters

Parameter	Meaning
client	(FTP_CLIENT *) Pointer to the FTP_Client structure.
path	(CHAR *) Pointer to the string containing the directory to remove.

Return Value

Upon successful completion of this service, it returns NU_SUCCESS. Otherwise, it returns the following:

```
FTP_STACK_ERROR,
FTP_SERVICE_UNAVAILABLE,
FTP_CMD_UNRECOGNIZED,
FTP_BAD_CMD_FORMAT,
FTP_CMD_NOT_IMPLEMENTED,
FTP_BAD_CMD_SEQUENCE,
FTP_INVALID_USER,
FTP_FILE_UNAVAILABLE or
FTP_BAD_RESPONSE.
```



NU_FCP_Client_CDUP

NU_FCP_Client_XCUP

```
INT FCP_Client_CDUP(FTP_CLIENT *client)
INT FCP_Client_XCUP(FTP_CLIENT *client)
```

This function provides a primitive to send an FTP CDUP (change directory to parent) message to the server. This function causes the server to change the working directory to the parent of the current directory. An error from the server indicates that the path was invalid or was inaccessible (due to privilege level).

Parameters

Parameter	Meaning
client	(FTP_CLIENT *) Pointer to the FTP_Client structure.

Return Value

Upon successful completion of this service, it returns NU_SUCCESS. Otherwise, it returns the following:

```
FTP_STACK_ERROR,
FTP_SERVICE_UNAVAILABLE,
FTP_CMD_UNRECOGNIZED,
FTP_BAD_CMD_FORMAT,
FTP_CMD_NOT_IMPLEMENTED,
FTP_INVALID_USER,
FTP_FILE_NOT_FOUND or
FTP_BAD_RESPONSE.
```

NU_FCP_Client_PASV

```
INT FCP_Client_PASV(FTP_CLIENT *client,  
                    struct addr_struct *servaddr)
```

This function provides a primitive to send an FTP PASV (passive connect) message to the server. An error will indicate that the server does not recognize or allow this command.

Parameters

Parameter	Meaning
client	(FTP_CLIENT *) Pointer to the FTP_Client structure.
servaddr	(STRUCT_ADDR_STRUCT *) Pointer to the structure containing the address to retrieve from the server.

Return Value

Upon successful completion of this service, it returns NU_SUCCESS. Otherwise, it returns the following:

```
FTP_STACK_ERROR,  
FTP_SERVICE_UNAVAILABLE,  
FTP_CMD_UNRECOGNIZED,  
FTP_BAD_CMD_FORMAT,  
FTP_CMD_NOT_IMPLEMENTED,  
FTP_INVALID_USER or  
FTP_BAD_RESPONSE.
```

NU_FCP_Client_QUIT

```
INT FCP_Client_QUIT(FTP_CLIENT *client)
```

This function provides a primitive to send an FTP QUIT message to the server. This function signals to the server that the client desires an end of the connection. The actual TCP close is initiated by the server.

Parameters

Parameter	Meaning
client	(FTP_CLIENT *) Pointer to the FTP_Client structure.

Return Value

Upon successful completion of this service, it returns NU_SUCCESS. Otherwise, it returns the following:

FTP_STACK_ERROR,
FTP_CMD_UNRECOGNIZED or
FTP_BAD_RESPONSE.

NU_FCP_Client_SYST

```
INT FCP_Client_SYST(FTP_CLIENT *client,  
                    CHAR *buffer,  
                    INT bufsize)
```

This function provides a primitive to send an FTP SYST (system information) message to the server.

Parameters

Parameter	Meaning
client	(FTP_CLIENT *) Pointer to the FTP_Client structure.
buffer	(CHAR *) Pointer to the storage location for the system information.
bufsize	(INT) Size of the buffer.

Return Value

Upon successful completion of this service, it returns NU_SUCCESS. Otherwise, it returns the following:

```
FTP_STACK_ERROR,  
FTP_SERVICE_UNAVAILABLE,  
FTP_CMD_UNRECOGNIZED,  
FTP_BAD_CMD_FORMAT,  
FTP_CMD_NOT_IMPLEMENTED,  
FTP_FILE_UNAVAILABLE or  
FTP_BAD_RESPONSE.
```



NU_FCP_Client_STAT

```

INT FCP_Client_STAT(FTP_CLIENT *client,
                    CHAR *path,
                    CHAR *buffer,
                    INT bufsize)

```

This function provides a primitive to send an FTP STAT (status) message to the server. The `path` parameter is a pointer to a character string containing a specification for the file(s) for which a status is requested. If empty, the request is for system information.

Parameters

Parameter	Meaning
<code>client</code>	(FTP_CLIENT *) Pointer to the FTP_Client structure.
<code>path</code>	(CHAR *) Pointer to the character string containing the path to the object whose status is desired.
<code>buffer</code>	(CHAR *) Pointer to the storage location for status information.
<code>bufsize</code>	(INT) Size of the buffer.

Return Value

Upon successful completion of this service, it returns `NU_SUCCESS`. Otherwise, it returns the following:

```

FTP_STACK_ERROR,
FTP_INVALID_USER,
FTP_SERVICE_UNAVAILABLE,
FTP_CMD_UNRECOGNIZED,
FTP_BAD_CMD_FORMAT,
FTP_CMD_NOT_IMPLEMENTED,
FTP_FILE_UNAVAILABLE or
FTP_BAD_RESPONSE.

```

NU_FCP_Client_HELP

```
INT FCP_Client_HELP(FTP_CLIENT *client,  
                    CHAR *topic,  
                    CHAR *buffer,  
                    INT bufsize)
```

This function provides a primitive to send an FTP HELP message to the server. The `topic` parameter is a pointer to a character string containing the topic for which help is requested. An empty string indicates a request for general help.

Parameters

Parameter	Meaning
<code>client</code>	(FTP_CLIENT *) Pointer to the <code>FTP_Client</code> structure.
<code>topic</code>	(CHAR *) Pointer to the character string containing the help topic.
<code>buffer</code>	(CHAR *) Pointer to the storage location for help information.
<code>bufsize</code>	(INT) Size of the buffer.

Return Value

Upon successful completion of this service, it returns `NU_SUCCESS`. Otherwise, it returns the following:

```
FTP_STACK_ERROR,  
FTP_SERVICE_UNAVAILABLE,  
FTP_CMD_UNRECOGNIZED,  
FTP_BAD_CMD_FORMAT,  
FTP_CMD_NOT_IMPLEMENTED,  
FTP_FILE_UNAVAILABLE or  
FTP_BAD_RESPONSE.
```



NU_FCP_Client_PWD

NU_FCP_Client_XPWD

```
INT FCP_Client_PWD(FTP_CLIENT *client, CHAR *buffer, INT bufsize)
INT FCP_Client_XPWD(FTP_CLIENT *client, CHAR *buffer, INT bufsize)
```

This function provides a primitive to send an FTP PWD (print working directory) message to the server. Note that the XPWD version is for backward compatibility with older servers that may not recognize the newer forms. The PWD variant will call the XPWD variant if an unrecognized command error is received. The XPWD function need not ever be used and is here for reference only.

Parameters

Parameter	Meaning
client	(FTP_CLIENT *) Pointer to the FTP_Client structure.
buffer	(CHAR *) Storage location of the directory name.
bufsize	(CHAR *) Size of the buffer.

Return Value

Upon successful completion of this service, it returns NU_SUCCESS. Otherwise, it returns the following:

```
FTP_STACK_ERROR,
FTP_SERVICE_UNAVAILABLE,
FTP_CMD_UNRECOGNIZED,
FTP_BAD_CMD_FORMAT,
FTP_CMD_NOT_IMPLEMENTED,
FTP_FILE_UNAVAILABLE or
FTP_BAD_RESPONSE.
```

NU_FCP_Client_NOOP

```
INT FCP_Client_NOOP(FTP_CLIENT *client)
```

This function provides a primitive to send an FTP NOOP (no operation) message to the server. This function is only included for RFC compliance, as it does not do anything of substance.

Parameters

Parameter	Meaning
client	(FTP_CLIENT *) Pointer to the FTP_Client structure.

Return Value

Upon successful completion of this service, it returns NU_SUCCESS. Otherwise, it returns the following:

```
FTP_STACK_ERROR,
FTP_SERVICE_UNAVAILABLE,
FTP_CMD_UNRECOGNIZED or
FTP_BAD_RESPONSE.
```

NU_FCP_Client_RNFR

```
INT FCP_Client_RNFR(FTP_CLIENT *client, CHAR *path)
```

This function provides a primitive to send an FTP RNFR (rename from) message to the server. An error from the server will indicate that the file is not accessible, the file does not exist, or that the command is not allowed at all. A success reply indicates that a RNTD command must be sent as the next command message.

Parameters

Parameter	Meaning
client	(FTP_CLIENT *) Pointer to the FTP_Client structure.
path	(CHAR *) Pointer to the string containing the original name.

Return Value

Upon successful completion of this service, it returns NU_SUCCESS. Otherwise, it returns the following:

```
FTP_STACK_ERROR,  
FTP_SERVICE_UNAVAILABLE,  
FTP_FILE_UNAVAILABLE,  
FTP_CMD_UNRECOGNIZED,  
FTP_BAD_CMD_FORMAT,  
FTP_CMD_NOT_IMPLEMENTED,  
FTP_INVALID_USER,  
FTP_FILE_UNAVAILABLE or  
FTP_BAD_RESPONSE.
```



NU_FCP_Client_RNTO

```
INT FCP_Client_RNTO(FTP_CLIENT *client, CHAR *path)
```

This function provides a primitive to send an FTP RNTO (rename to) message to the server. This function will return a failure if it is not immediately preceded by a RNFR command.

Parameters

Parameter	Meaning
client	(FTP_CLIENT *) Pointer to the FTP_Client structure.
path	(CHAR *) Pointer to the string containing the new name.

Return Value

Upon successful completion of this service, it returns NU_SUCCESS. Otherwise, it returns the following:

```
FTP_STACK_ERROR,  
FTP_SERVICE_UNAVAILABLE,  
FTP_CMD_UNRECOGNIZED,  
FTP_BAD_CMD_FORMAT,  
FTP_CMD_NOT_IMPLEMENTED,  
FTP_BAD_CMD_SEQUENCE,  
FTP_INVALID_USER,  
FTP_NEED_ACCOUNT,  
FTP_INVALID_FILE_NAME or  
FTP_BAD_RESPONSE.
```



NU_FCP_Client_DELE

```
INT FCP_Client_DELE(FTP_CLIENT *client, CHAR *path)
```

This function provides a primitive to send an FTP DELE message to the server. The `path` parameter is a pointer to a character string containing the name of a remote file to delete. If a path is included in the string, it refers to a path on the server machine. An error indicates that the file did not exist or that the file could not be deleted.

Parameters

Parameter	Meaning
<code>client</code>	(FTP_CLIENT *) Pointer to the FTP_Client structure.
<code>path</code>	(CHAR *) Pointer to the string containing the file to delete.

Return Value

Upon successful completion of this service, it returns `NU_SUCCESS`. Otherwise, it returns the following:

```
FTP_STACK_ERROR,
FTP_SERVICE_UNAVAILABLE,
FTP_FILE_UNAVAILABLE,
FTP_CMD_UNRECOGNIZED,
FTP_BAD_CMD_FORMAT,
FTP_CMD_NOT_IMPLEMENTED,
FTP_INVALID_USER,
FTP_FILE_UNAVAILABLE or
FTP_BAD_RESPONSE.
```



3

Nucleus Telnet

Files Shipped with Nucleus
Telnet

Defines and Data Structures

Service Call Primitives

Internal Service Calls



Files Shipped with Nucleus Telnet

File Name	Description
TL_UTIL.C	Telnet utilities.
NEGOTIAT.C	Telnet negotiation utilities.
DBC.C	Core Nucleus Debugger routines.
DBT.C	Nucleus Debugger routines.
DB_DEFS.H	Nucleus Debugger external function reference.
DB_EXTR.H	Nucleus Debugger Constants.
VTKEYS.H	Terminal emulation key mapping table.
TELOPTS.H	Macros for Telnet codes.
TEL_EXTR.H	External definitions for Telnet functions.
NVT.H	Negotiation options table for client and server.
NKEYS.H	The key values of the PC keystrokes.
N_ANSI.H	Macros for Non-ANSI functions.
NU_TLNET.H	This is the only file needed to include by an application to access all User-Level Telnet functions.
WINDAT.H	Data structure of the parameters for a Telnet session.

Defines and Data Structures

Defines

Following are configurable constants for the Nucleus Telnet Server.

Define Variable	Value
MAX_NEGO_OPTIONS	5
MAX_NEGO_LENGTH	MAX_NEGO_OPTIONS * 4
MAX_TABLE_ENTRIES	MAX_NEGO_OPTIONS * 2
N_COLORS	4

TN_VERSION_COMP

To make NU_Telnet_Send() service to operate as it did in the previous release, change the compatibility macro TN_VERSION_COMP in TELNET/INC/TELOPTS.H to TN_1_1 #define TN_VERSION_COMP TN_1_1.



Data Structures

The data structure `TN_SESSION_STRUCT` contains all parameters for a Telnet Session.

```
struct TN_SESSION_STRUCT
{
    unsigned short colors[NCOLORS];
    unsigned char mname[16],
        linemode_mask,
        linemode[82];
    int pnum,
        socket,
        bkscroll,
        width,
        rows,
        telstate,
        substate,
        termstate,
        nego_NAWS,
        crfollow,
        nego_TERMTYPE,
        bksp,
        del,
        slc[31],
        slm[31];
    char *ftppopts;
    unsigned int mapoutput:1,
        vtwrap:1,
        lmflag:1,
        lmedit:1,
        litflag:1,
        litecho:1,
        softtab:1,
        trapsig:1,
        halfdup:1,
        termsent:1,
        ibinary:1,
        iwantbinary:1,
        ubinary:1,
        uwantbinary:1,
        igoahead:1,
        ugoahead:1,
        echo:1,
        timing:1,
        capon:1,
        condebug:2;
    FILE *capfp;
    struct TN_SESSION_STRUCT *next, *prev;
};
```



The data structure `NVT_KEY` is the data structure for the mapping table.

```
typedef struct nvt
{
    int key_value;
    char key_string[5];          /* the NVT ESC string */
    char string2[6];
} NVT_KEY;
```

Service Call Primitives

The Nucleus Telnet API service call primitives are simple functions that handle the task of initializing Telnet applications, negotiating with the other side and building the connection.

NU_Telnet_Server_Accept

```
INT NU_Telnet_Server_Accept(INT socket)
```

This function is responsible for accepting a connection from a Telnet client.

Parameters

Parameter	Meaning
socket	(INT) Specifies the socket index of this connection.

Return Value

Upon successful completion of this service, it returns a socket index. Otherwise, it returns a value less than 0.



NU_Telnet_Send

```
INT NU_Telnet_Send(INT socket, CHAR *fmt)
```

This function is responsible for sending data to the network during Telnet communication. `NU_Telnet_Send` is different from `NU_Send()` in that a variable number of arguments are allowed. If this feature is not needed, `NU_Send` should be used instead.

Parameters

Parameter	Meaning
socket	(INT) Specifies the socket index of this connection.
fmt	(CHAR *) pointer to the buffer containing the data to be sent.

Return Value

Upon successful completion of this service, it returns the number of bytes sent. Otherwise, it returns a value less than 0.



NU_Telnet_Start_Negotiate

```
VOID NU_Telnet_Start_Negotiate(INT socket, CHAR *nego_table)
```

This function is responsible for starting a Telnet negotiation.

Parameters

Parameter	Meaning
socket	(INT) Specifies the socket index of this connection.
nego_table	(CHAR *) A pointer to an array of even elements, the first element of each pair is the index of the Telnet negotiation command, the second is the Telnet negotiation option.

Return Value

None



NU_Telnet_Specific_Negotiate

```
VOID NU_Telnet_Specific_Negotiate(INT socket, INT option)
```

This function is responsible for insisting on a negotiation for window size (terminal screen) until it gets a response from the other end. The Telnet service `NU_Telnet_Start_Negotiate` is designed to do a basic negotiation. It is easy for the user to modify this function by following its style to require specific negotiation for some other parameter. Its main body is a while loop, as follows:

```
while (1)
{
    ...
    cnt = NU_Recv(socket, data, 200, 0);
    ...
    NU_Telnet_Parse(socket, data, cnt, 0);
    ...
    if (!(tw->nego_NAWS&NEGO_HE_ACKED))
        NU_Telnet_Send(socket, "%c%c%c", IAC, DOTE, NAWS);
    else
        break;

    NU_Sleep(1);
}
```

Where `nego_NAWS` is a global variable used as a flag for the negotiation status of NAWS (the window size). `nego_NAWS` is declared inside `WINDAT.H`, and `NEGO_HE_ACKED`, `IAC`, `DOTE` and `NAWS` are macro defined inside `TELOPTS.H`. For further information please refer to RFC 854.

Parameters

Parameter	Meaning
socket	(INT) The socket index of this connection.
option	(INT) The telnet option for this connection.

Return Value

None



NU_Install_Negotiate_Options

```
VOID NU_Install_Negotiate_Options(CHAR *nego_buf, CHAR *nego_table)
```

This function is responsible for installing the negotiation options. It is called by `NU_Telnet_Start_Negotiation()`. This function does not need to be called directly. `NU_Install_Negotiate_Options` will install all the negotiation options into the buffer according to the command and option tables. These tables, called `client_nego_table[]` and `server_nego_table[]`, are declared in the file `NVT.H`. The server table is declared as follows:

```
unsigned char server_nego_table[MAX_TABLE_ENTRIES]={
    NOTHING, BINARY,
    WILL,     ECHO,
    DO_WILL,  SGA,
    DO,       NAWS,
    DO,       TERMTYPE
};
```

The first column lists the commands, the second lists the options. The user can find these MACROS of commands and of options in `TELOPTS.H`. More information on Telnet options, option formatting, and the definition of each command can be found in RFCs 854 and 855.

The table `nego_table[]` is designed for basic usage. This table can be modified to match individual needs. However, the formatting should be followed exactly.

Parameters

Parameter	Meaning
<code>nego_buf</code>	(unsigned char *) Pointer to the buffer where the negotiation options are to be installed.
<code>nego_table</code>	(CHAR *) Pointer to the table of negotiation options to install.

Return Value

None



NU_Telnet_Init_Parameters

```
VOID NU_Telnet_Init_Parameters(INT socket)
```

This function is responsible for initializing the parameters of a Telnet session. The memory for a Telnet session structure is allocated when its fields are initialized. The array of pointers to Telnet sessions is declared in `WINDAT.H`.

The socket index of this connection is specified by `socket`.

Parameters

Parameter	Meaning
<code>socket</code>	(INT) The socket associated with the connection.

Return Value

None



NU_Telnet_Free_Parameters

```
VOID NU_Telnet_Free_Parameters( INT socket )
```

This function is responsible for freeing the memory allocated by `NU_Telnet_Init_Parameters()`. These two functions must be used together.

The socket index of this connection is specified by `socket`.

Parameters

Parameter	Meaning
<code>socket</code>	(INT) The socket associated with the connection.

Return Value

None



NU_Telnet_Check_Connection

```
INT NU_Telnet_Check_Connection(INT socket, INT close_it)
```

This function is responsible for checking and closing a Telnet connection.

Parameters

Parameter	Meaning
socket	(INT) Specifies the socket index of this connection.
close_it	(INT) Specifies if the service should check the connection only, or if a close should be performed.

Return Value

Upon successful completion of this service, it returns SCLOSED. Otherwise it returns a value less than 0.



NU_Telnet_Client_Connect

```
INT NU_Telnet_Client_Connect(CHAR *server_ip, CHAR *server_name)
```

This function is responsible for opening a connection with a Telnet server.

Parameters

Parameter	Meaning
server_ip	(CHAR *) Pointer to the network ip address of the server host.
server_name	(CHAR *) Pointer to the machine name of the server host. Nucleus Telnet utilities does not use the machine table.

Return Value

Upon successful completion of this service, it returns the socket index. Otherwise, it returns a value less than 0.



NU_Received_Exit

```
INT NU_Received_Exit(CHAR *data, INT cnt);
```

This function validates the exit command from the client.

Parameters

Parameter	Meaning
data	(CHAR *) Pointer to the input string to be parsed.
cnt	(INT) The length of the string to be parsed.

Return Value

This function returns 0 if no exit command was found and 1 if an exit command was found.



NU_Close_And_Check_Retval

```
INT NU_Close_And_Check_Retval( INT socket )
```

This function validates the return value of `NU_Close_Socket()`.

Parameters

Parameter	Meaning
socket	(INT) The socket index of the Telnet connection.

Return Value

This function returns the status of the `NU_Close_Socket()` call.



NU_Telnet_Socket

```
INT NU_Telnet_Socket (CHAR *server_ip, CHAR *server_name)
```

This function creates the first socket for the telnet connection.

Parameters

Parameter	Meaning
server_ip	(CHAR *) Pointer to the Telnet server ip.
server_name	(CHAR *) Pointer to the Telnet server name.

Return Value

Upon successful completion of this service, it returns the socket index. Otherwise, it returns -1.



Internal Service Calls

The Nucleus Telnet API internal service calls are functions that handle the details of the communication according to the Telnet protocol (see RFC 854). The functions discussed in this section include a brief description, the function's syntax, a description of any parameters used and return values.

NU_Telnet_Pick_Up_Ascii

```
INT NU_Telnet_Pick_Up_Ascii(UINT8 *st, INT cnt)
```

This function is responsible for picking up only ASCII code bytes from the incoming data. It will filter out all Telnet commands and options, terminal emulation key codes, and ESC characters. It collects the rest of the data in the original buffer.

Parameters

Parameter	Meaning
st	(unsigned char *) Pointer to the data buffer.
cnt	(INT) Specifies how many bytes of data are in the buffer pointed to by st.

Return Value

Upon successful completion of this service, it returns NU_SUCCESS. Otherwise it returns one of the following codes:

Code	Meaning
1	Sub-negotiation codes
2	Telnet command and option
4	ASCII codes
8	ESC string



NU_Telnet_Parse

```
INT NU_Telnet_Parse(INT socket, UINT8 *st, INT cnt,  
                    INT terminal)
```

This function is responsible for parsing the incoming data from the other end of a Telnet connection. It will recognize the Telnet commands and options and execute them automatically.

Parameters

Parameter	Meaning
Socket	(INT) The socket index of this connection.
St	(unsigned char *) Pointer to the buffer where the received data is stored.
Cnt	(INT) The length of the data buffer.
Terminal	(INT) Specifies whether data other than Telnet commands and telnet options will be automatically dumped to the users terminal screen when one is present.

Return Value

This service always returns NU_SUCCESS.



NU_Telnet_Get_Filtered_Char

```
CHAR NU_Telnet_Get_Filtered_Char(INT socket, INT block)
```

This function is responsible for getting one character from the network. If the incoming byte is the first byte of the string of a Telnet command or option, or is the string of terminal emulation keystroke code, this function will continue to collect characters. The resulting string is then passed to the appropriate function for processing. If the parameter block is set, this service will wait until it gets one character, and will return it to the caller. If the parameter block is not set, it simply checks the network buffer without waiting.

Parameters

Parameter	Meaning
socket	(INT) Specifies the socket index of this connection.
block	(INT) Specifies whether or not the routine needs to wait for incoming data.

Return Value

Upon successful completion of this service, it returns an ASCII character. If block equals 0, and there is no incoming data, the return value is 0.



NU_Receive_NVT_Key

```
INT NU_Receive_NVT_Key(INT socket, CHAR *string, CHAR execute)
```

This function is responsible for converting a received key mapping code to the corresponding local key value (according to the pre-defined mapping table declared in VTKEYS.H), and explaining it, or executing the function that it represents. The Nucleus NET Telnet Utilities cannot be designed to cover all terminal types. Local key values are dependent on the user's device and application program. This table can be modified, expanded, or more tables can be added to suit the environment. The user may need to add function calls to do what each specific key will invoke.

Parameters

Parameter	Meaning
socket	(INT) Specifies the socket index of this connection.
string	(CHAR *) Pointer to the key code string to be converted.
execute	(CHAR) Flag that determines if the function that the key value represents is executed.

Return Value

Upon successful completion of this service, it returns the number of bytes `string` contains. Otherwise, it returns 0.



NU_Send_NVT_Key

```
VOID NU_Send_NVT_key(INT socket, unsigned int c)
```

This function is responsible for sending a local keystroke to the remote host. If the local key value is listed in the mapping table pre-defined in `VTKEYS.H`, it will be converted to the terminal emulation key string according to the table. Then the string will be sent to the remote host. Nucleus NET Telnet Utilities cannot be designed to cover all terminal types. Local key values are dependent on the user's device and how the user's application program produces it. This table can be modified, expanded, or more tables can be added according to the environment.

Parameters

Parameter	Meaning
socket	(INT) Specifies the socket index of this connection.
c	(unsigned int) The value of local key.

Return Value

None



4

Nucleus TFTP Server

Introduction

Files Shipped with Nucleus TFTP
Server

TFTP Server Defines and Data
Structures

TFTP Server Initialization



Introduction

The Nucleus TFTP Server supports OCTET (binary) mode, as discussed within RFC 1350. It processes read and write requests from a TFTP Client. The TFTP Server, depending on the request, will read data from or store data into the user specified file system via function calls to the Nucleus FAL (File Abstraction Layer). The Nucleus FAL supports Nucleus File, the NT File System and an In-Memory File System, or it can be easily modified to meet the user's current file system.

Files Shipped with Nucleus TFTP Server

File Name	Description
TFTPS.C	All TFTP routines necessary to implement an RFC compliant TFTP Server.
NU_TFTPS.H	This is the only file needed to include by an application to access all User-Level Nucleus TFTP Server functions.
TFTPEXTS.H	Function prototypes of all TFTP Server routines.



TFTP Server Defines and Data Structures

Defines

The TFTP Server uses several constants in the form of `#defines`. The following is a list of these defines, their values, and in what category they belong (whether they are a return code, internally used values, or parameter values).

Name	Value	Category
TFTP_RRQ_OPCODE	1	Parameter Value
TFTP_WRQ_OPCODE	2	Parameter Value
TFTP_DATA_OPCODE	3	Parameter Value
TFTP_ACK_OPCODE	4	Parameter Value
TFTP_ERROR_OPCODE	5	Parameter Value
TFTP_OACK_OPCODE	6	Parameter Value
TFTP_ERROR	-302	Return Code
TFTP_FILE_NFOUND	-303	Return Code
TFTP_ACCESS_VIOLATION	-304	Return Code
TFTP_DISK_FULL	-305	Return Code
TFTP_FILE_ERROR	-306	Return Code
TFTP_BAD_OPERATION	-307	Return Code
TFTP_UNKNOWN_TID	-308	Return Code
TFTP_FILE_EXISTS	-309	Return Code
TFTP_NO_SUCH_USER	-310	Return Code
TFTP_BAD_OPTION	-311	Return Code
TFTP_NO_MEMORY	-312	Return Code
TFTP_CON_FAILURE	-313	Return Code
TFTP_SUCCESS	0	Return Code
TRANSFERRING_FILE	100	Internally Used Value
TRANSFER_COMPLETE	102	Internally Used Value
TFTP_BLOCK_SIZE_DEFAULT	512	Internally Used Value
TFTP_HEADER_SIZE	4	Internally Used Value
TFTP_ACK_SIZE	4	Internally Used Value
TFTP_BLOCK_SIZE_MAX	65464	Internally Used Value
TFTP_BLOCK_SIZE_MIN	8	Internally Used Value
TFTP_BUFFER_SIZE_MIN	50	Internally Used Value
TFTP_PARSING_LENGTH	128	Internally Used Value
READ_TYPE	1	Internally Used Value
WRITE_TYPE	2	Internally Used Value
TFTP_NUM_RETRANS	3	Internally Used Configurable Value
TFTP_TIMEOUT_DEFAULT	60	Internally Used Configurable Value



Data Structures

The TFTP_OPTIONS data structure stores the timeout, blksize and tsize as specified by the user (or the default value if not specified). The structure also contains the flags timeout_acknowledged, blksize_acknowledged, and tsize_acknowledged.

```
typedef struct tftp_options
{
    UINT16  timeout;
    UINT16  blksize;
    UINT32  tsize;
    INT16   timeout_acknowledged;
    INT16   blksize_acknowledged;
    INT16   tsize_acknowledged;
} TFTP_OPTIONS;
```

The TFTP_CB data structure serves as the TFTP control block.

```
typedef struct tftp_cb
{
    CHAR      *trans_buf;
    TFTP_OPTIONS options;
    UINT32     cur_buf_size;
    Struct     addr_struct server_addr;
    INT16      socket_number;
    UINT16     block_number;
    INT16      tid;
    INT16      status;
    INT16      type;
    char       *file_name;
    char       *mode;
} TFTP_CB;
```



TFTP Server Initialization

TFTP_Server_Init

```
STATUS TFTP_Server_Init(CHAR *dv_name)
```

This function initializes the TFTP_Server_Task. This is the external interface to start the TFTP service within the user's application. The parameter dv_name is passed in after the initialization of the user's network devices. The user may specify which device to use or pass in NULL to use any available device.

Parameters

Parameter	Description
dv_name	The name of the device that will hold the TFTP Server or NULL if the user wishes to not use a specific device.

Return Value

Upon successful completion of this service, it returns NU_SUCCESS. Otherwise, it returns NU_INVALID_POOL, NU_NO_MEMORY or NU_INVALID_SIZE.





5

Demonstration Application

Configuring Nucleus Extended Protocol
Package Demonstration



Configuring Nucleus Extended Protocol Package Demonstration

This chapter provides generic information about the Nucleus Extended Protocol Package Demonstration program. This application actually consists of four demos: A TFTP Server, a Telnet Server, an FTP Server, and an FTP Client. The application can be built to include one, all, or any combination of the aforementioned demos. Each of these is discussed below.

Note that FTP and TFTP require a file system to work. The Extended Protocol Package makes use of the File Abstraction Layer provided with Nucleus NET for this purpose. FAL includes support for an In-Memory File System so the demo can be executed even if a physical storage device is not present. Please see the FAL documentation for more information.

The demonstration application provided with Nucleus NET can be executed in a *lite* mode utilizing only the Nucleus NET loopback interface. The Extended Protocol Package demonstration cannot be used in this manner. A physical device such as Ethernet and/or PPP must be available, and there must be a working driver for this interface.

The package demonstration consists of six files; `DBC.C`, `DBT.C`, `DB_DEFS.H`, `DB_EXTR.H`, `DEMOI.C`, and `XPROTD.C`. An explanation of the functionality of each file is given below:

File	Functionality
<code>DBC.C</code>	Core Nucleus Debugger routines used by the Telnet demo task.
<code>DBT.C</code>	Nucleus Debugger routines used by the Telnet demo task.
<code>DB_DEFS.H</code>	Nucleus Debugger constants.
<code>DB_EXTR.H</code>	Nucleus Debugger routine declarations.
<code>DEMOI.C</code>	General network initialization for the demonstration. Also includes user configurable parameters.
<code>XPROTD.C</code>	Demonstration application.



DEMOI.C

The DEMOI.C file is the target specific file for the demonstration program and is provided with Nucleus NET. It performs network initialization and device initialization. Please see the Nucleus NET Reference Manual for more information on this file.

Once DEMOI_INIT_TASK has completed, an event is triggered to begin the tasks in the XPROTD.C file.

Following are the three global variables you may need to configure in DEMOI.C, this is usually done during software installation. The FTP server and local IP address can be hard coded into DEMOI.C, the default configuration, or NU_GET_HOST_BY_NAME can be used to resolve the FTP server address and DHCP or BOOTP can be used to resolve the local IP address. The three IP addresses must be valid before DEMOI.C sets the event to start the Nucleus Extended Protocol demonstration.

Global	Description
DEMOI_FTP_SERVER	The IP address of the remote FTP Server with which the FTP Client in the demonstration program communicates.
DEMOI_LOCAL_IP	The local IP address of the device running the demonstration program.
DEMOI_SUBNET	The subnet mask of the local IP address.

XPROTD.C

The XPROTD.C file that came with your port of the Nucleus Extended Protocol Package provides a demonstration of how to incorporate each of these protocols into your own application. XPROTD.C is responsible for the following functionality: initializing application tasks, initializing the system memory pool, initializing the File System, and initializing the Nucleus TFTP Server, Telnet Server, FTP Server and FTP Client.

The demonstration consists of three tasks. Each task begins upon completion of DEMOI_INIT_TASK in the DEMOI.C file. Completion of this task sets a Nucleus PLUS event. The tasks that make up the extended protocol demonstration will suspend pending the setting of this event. A description of each protocol's behavior and configurable parameters follows.



FTP Client

The FTP Client can be included / excluded by flipping the switch `FTPC_DEMO` found at the top of `XPROTD.C`. In order to execute the FTP Client portion of the demonstration, there must be a FTP Server on your network with which the FTP Client can communicate. The task entry function for the FTP Client is `XPR_FTPC_TASK`. It will also be necessary to configure the username and password used by the client to log onto the server. Modify the two global variables `XPR_FTPC_PASW` and `XPR_FTPC_USNAME` found in `XPROTD.C` to configure the username and password. The IP address of the server with which the FTP Client will communicate is defined by the global variable `DEMOI_FTP_Server` found in `DEMOI.C`.

When the FTP Client executes, it will create the file `DEMO.TXT`, open a data connection with a remote server, login with a username and password, create a new directory `TESTDIR`, and change to the new directory `TESTDIR`. The task will then transmit the file `DEMO.TXT` to the server, retrieve the file `DEMO.TXT` from the server, change to the parent directory, retrieve a directory listing of `TESTDIR`, delete the file `DEMO.TXT` and delete the directory `TESTDIR` for as many iterations as the user has indicated in `XPR_FTPC_LOOPS`.

User configurable parameters are located in the section entitled `USER CONFIGURABLE PARAMETERS` and consist of the following:

Parameter	Description
<code>XPR_FTPC_PASW</code>	User name to log onto the FTP Server.
<code>XPR_FTPC_USNAME</code>	User password to log onto the FTP Server.
<code>XPR_FTPC_SLP</code>	Ticks to sleep between FTP Client commands.
<code>XPR_FTPC_LOOPS</code>	Iterations of FTP Client send/retrieve.
<code>XPR_FTPC_FILE_SIZE</code>	Size of file for FTP Client to create.



FTP Server

The FTP Server can be included / excluded by flipping the switch `FTPS_DEMO` found at the top of `XPROTD.C`. The task entry function for the FTP Server is `XPR_Init`. The IP address of the server is defined by the global variable `DEMOI_LOCAL_IP` found in `DEMOI.C`. The FTP Server is dependent on a real file system, as opposed to the In-Memory File system. Nucleus FILE can be used to meet the file system requirements of the FTP Server and if not present can be purchased from ATI. See the FAL chapter in the *Nucleus NET Reference Manual* for more information on how to configure the file system.

When the FTP Server executes, it will wait indefinitely for a connection from an FTP Client on the default port 21. Once a connection is received, it will process the connection according to the client's requests.

An FTP Client is required to communicate with the Nucleus FTP Server. The FTP Client utility supplied with Windows NT/95/98 and the various flavors of UNIX and Linux can be used for this purpose. If the IP address of the embedded target FTP Server is 192.200.100.1, then simply enter "**ftp 192.200.100.1**" at a command prompt.

TFTP Server

The TFTP Server can be included / excluded by flipping the switch `TFTPS_DEMO` found at the top of `XPROTD.C`. The task entry function for the TFTP Server is `XPR_Init`. The IP address of the server is defined by the global variable `DEMOI_LOCAL_IP` found in `DEMOI.C`.

When the TFTP Server executes, it will wait indefinitely for a connection from a TFTP Client on the default port 69. Once a connection is received, it will process the connection according to the client's requests.

A TFTP Client is required to communicate with the Nucleus TFTP Server. A TFTP Client utility is generally supplied with Windows NT and the various flavors of UNIX and Linux. For other platforms you will need to acquire a TFTP Client. Shareware and/or freeware clients can be found on the Internet. To communicate with the TFTP server simply enter "**tftp xxx.xxx.xxx.xxx**", where `xxx.xxx.xxx.xxx` is the IP address of the embedded target.



Telnet Server

The Telnet Server can be included / excluded by flipping the switch `TELNET_DEMO` found at the top of `XPROTD.C`. The task entry function for the Telnet Server is `XPR_Telnet_Task`. The IP address of the server is defined by the global variable `DEMOI_LOCAL_IP` found in `DEMOI.C`.

When the Telnet Server executes, it will begin a Nucleus Debug Session and wait indefinitely for a connection from a Telnet Client on the default port 23. Once a connection is received, it will process the connection according to the client's requests.

A Telnet Client is required to communicate with the Nucleus Telnet server. The Telnet Client utility supplied with Windows NT/95/98 and the various flavors of UNIX and Linux can be used for this purpose. If the IP address of the embedded target is 192.200.100.1, then enter "**telnet 192.200.100.1**" at a command prompt.





Appendix

Enhancements and Nucleus
Extended Protocol API Changes



Extended Protocol Package 1.2

Enhancements

- Simple User Authentication – Can be able to add a user and edit password for the FTP Server.
- Able to send time and date when doing a DIR listing on the FTP Server.
- FTP Server able to operate in passive mode.
- Error handling has been improved.

New API Calls

Name	Description
NU_FTPC_Client_Append_To_File	This function appends the local file to a remote file that exists on the FTP server.
NU_FTPC_Client_Nlist	This function retrieves a directory listing from the FTP server.

New Files

Filename	Description
FC.C	FTP Client and Server common functions. These functions can be used by both client and server.
FC_EXTR.H	Function declarations for FC.C.
FTP_CFG.C	FTP Client and Server configuration file for customization of the FTP embedded application by the developer. This file contains initialized data to be compiled in the project.
FTP_CFG.H	FTP Client and Server configuration file for customization of the FTP embedded application by the developer. This file contains constant definitions.

Newly Defined Macros

```

FTPS_INACT_TIMEOUT
FTPS_DATA_TIMEOUT
FTPS_GET_CURRENT_YEAR
FTP_MAX_ID_LENGTH
FTP_MAX_PW_LENGTH
FTPC_INACT_TIMEOUT
FTPC_CCONN_TIMEOUT
FTPC_DATACT_TIMEOUT

```

